VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

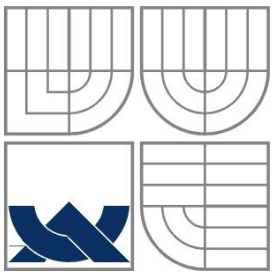
MODULÁRNÍ NÁSTROJ PRO TVORBU HER S UMĚLOU INTELIGENCÍ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

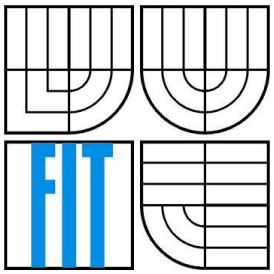
AUTOR PRÁCE
AUTHOR

JAN KLIKA

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

MODULÁRNÍ NÁSTROJ PRO TVORBU HER S UMĚLOU INTELIGENCÍ

MODULAR TOOL FOR CREATING GAMES WITH ARTIFICIAL INTELLIGENCE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAN KLIKA

VEDOUCÍ PRÁCE
SUPERVISOR

ING. JAN KOUŘIL

BRNO 2012

Abstrakt

Tato bakalářská práce se zabývá návrhem a implementací nástroje pro usnadnění tvorby umělých inteligencí. Nástroj používá vlastní univerzální komunikační protokol, podporuje více typů her, je rozšiřitelný o další hry pomocí dynamických knihoven, podporuje zobrazení ladících informací z umělé inteligence a je lokalizovaný do českého a anglického jazyka.

Abstract

This bachelor thesis describes the design and implementation of tool for facilitate the creation of artificial intelligence. The tool uses its own universal communication protocol, supports multiple types of games, is extensible for other games using dynamic libraries, supports the display of debugging information of artificial intelligence and is localized in English and Czech.

Klíčová slova

Umělá inteligence, komunikační protokol, modulární, multiplatformní, hry, Qt

Keywords

Artificial intelligence, communication protocol, modular, multiplatform, games, Qt

Citace

Klika Jan: Modulární nástroj pro tvorbu her s umělou inteligencí, bakalářská práce, Brno, FIT VUT v Brně, 2012

Modulární nástroj pro tvorbu her s umělou inteligencí

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Jana Kouřila. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……

Jan Klika
11.5.2012

Poděkování

Rád bych poděkoval Ing. Janu Kouřilovi za zajímavý námět a bezproblémovou spolupráci při tvorbě této práce.

Obsah

1 Úvod

Hry patří k člověku odjakživa. Sloužily především pro zábavu a srovnávání jedinců jak na fyzické, tak inteligenční úrovni. S rozvojem počítačů a jejich dostupností se spousta logických her převedla také do elektronické podoby, kde je možné utkat se s počítačovým soupeřem s tzv. „umělou“ inteligencí.

Umělá inteligence bývá často oddělena od grafické části hry, což umožňuje oddělený vývoj umělých inteligencí, použití jednoho grafického rozhraní pro více umělých inteligencí a v neposlední řadě porovnání umělých inteligencí mezi sebou. Pro komunikaci umělé inteligence a grafické části hry se používá textový protokol, který je navržený na míru konkrétní hře.

Cílem této práce je vytvořit nástroj, který by zájemcům o tvorbu umělých inteligencí usnadnil jejich vývoj. Tento nástroj bude obsahovat různé hry, pro které bude možné umělé inteligence vyvíjet. Komunikace s umělými inteligencemi bude probíhat pomocí nově navrženého univerzálního komunikačního protokolu, díky čemuž odpadne nutnost studovat pro každou hru komunikační protokol, který se u ní běžně používá. Implementované hry však mohou podporovat více protokolů, například pro porovnání vytvořené umělé inteligence s již existujícími, které používají jiné, jednoúčelové komunikační protokoly.

2 Používané komunikační protokoly

V této kapitole se zaměříme na komunikaci externích umělých inteligencí (brainů) s aplikací zprostředkující samotnou hru.

Možností komunikace umělé inteligence s aplikací je více. Používá se například komunikace přes soubory, roury nebo TCP/IP. My se budeme zabývat nejpoužívanějším způsobem, a to komunikací přes roury. Roura je nástroj pro spojování procesů v řetězce pomocí vzájemného propojení standardních proudů tak, že vždy výstup procesu (stdout) je nasměrován přímo do vstupu (stdin) následujícího procesu. Jde tedy o jednosměrnou meziprocesovou komunikaci (IPC), pro jejíž vytvoření se používá fronta FIFO, která se vytváří u příbuzných procesů a automaticky zaniká po jejich ukončení. Tento způsob komunikace je pro umělé inteligence výhodný z důvodu rychlosti komunikace a snadné implementace. Brain komunikuje přes standardní vstup a výstup, nemusí tedy vědět nic o rourách a chová se, jakoby četl z klávesnice a zapisoval na obrazovku. Díky tomu může být brain vytvořen prakticky v jakémkoliv programovacím jazyce, který umožňuje vytvářet konzolové aplikace.

Nyní se podíváme na ukázku tří používaných komunikačních protokolů založených právě na komunikaci pomocí rour. Prvním je protokol používaný v piškvorkovém manažeru Petra Laštovičky. Další dva protokoly jsou standardizované protokoly pro umělé inteligence hry šachy.

2.1 Protokol pro piškvorky

Tento protokol je používán v programu Piškvorky od Petra Laštovičky. Tento program funguje jako piškvorkový manažer, tedy program, který zprostředkovává zápas. Tento program umožňuje jak hru dvou fyzických hráčů proti sobě, tak hru fyzického hráče proti umělé inteligenci. Největší předností je však možnost hry dvou umělých inteligencí proti sobě, což umožňuje jejich porovnání. Tento program je používán pro pořádání každoročního mezinárodního turnaje umělých inteligencí piškvorek na serveru http://gomocup.wz.cz.

Jak již bylo zmíněno výše, protokol funguje pomocí rour. Jde o jednoduchý protokol navržený pouze pro hru piškvorky. Jednotlivé příkazy jsou zasílány vždy každý na samostatném řádku a měly by být psány velkými písmeny, jejich parametry pak malými písmeny. Příkazy tohoto protokolu se dají rozdělit na povinné, nepovinné a příkazy zasílané umělou inteligencí.

Mezi povinné příkazy se řadí takové, které jsou nutné pro samotný běh hry. Jsou to příkazy pro nastavení velikosti hrací plochy a start nové hry, dále pro předání tahu mezi umělou inteligencí a grafickou částí hry a příkaz umělé inteligenci k zahájení hry jako první hráč. Patří sem také příkazy pro vnucení herní plochy pro pokračování v rozehrané partii, pro předání dalších informací umělé inteligenci a pro získání informací o umělé inteligenci, jako název brainu a informace o autorovi. Posledním z povinných příkazů je příkaz k ukončení umělé inteligence.

Nepovinné příkazy slouží k rozšíření možností hry, jako například hra na obdélníkové hrací ploše, restart hry se stejným nastavením, vrácení provedeného tahu zpět nebo vnucení jiného tahu.

Všechny výše zmíněné příkazy jsou zasílány grafickou částí hry a na většinu z nich umělá inteligence nějak odpovídá. Jsou zde však také příkazy, které zasílá sama umělá inteligence. Jde především o zasílání různých zpráv, ať již chybových hlášení, ladících informací pro autora nebo běžné zprávy pro uživatele.

Podrobnější popis protokolu je uveden v "Příloha A - Protokol pro piškvorky".

Čerpáno z [1].

2.2 Chess Engine Cominucation Protocol

Chess Engine Communication Protocol je protokol navržený Timem Mannem, autorem XBoard. Tento protokol byl vytvořen pro komunikaci grafického rozhraní hry šachy - XBoard, a umělou inteligencí této hry - GNU Chess. Nyní je podporován velkým množstvím šachových brainů a grafických rozhraních.

Je to textový protokol s příkazy na samostatných řádcích. Vzhledem k tomu, že je vytvořen přímo pro hru šachy, má spoustu příkazů pokrývajících veškeré možnosti této hry. Jeho nevýhodou je však již zmíněný velký počet příkazů, které je nutné implementovat pro jeho použití. Z toho důvodu byl vytvořen další protokol pro komunikaci šachových programů s umělou inteligencí, jehož snahou bylo zjednodušení komunikačního protokolu, a tím i jeho implementace. Jedná se o UCI, tedy Universal Chess Interface, se kterým se seznámíme v další podkapitole.

Tento protokol se postupně vyvíjí a s novými verzemi vznikají nové příkazy a možnosti, některé se naopak přestávají používat. Kvůli zpětné kompatibilitě je však potřeba implementovat všechny tyto příkazy, jejichž množství se s novými verzemi zvětšuje. Současnou verzí protokolu je verze 2f.

Oproti protokolu Petra Laštovičky obsahuje tento protokol mezi příkazy také ověření typu použitého protokolu a jeho používané verze. Pochopitelně pak také obsahuje příkazy pro spuštění nové hry, předávání tahů, předávání chybových hlášení a ukončení umělé inteligence. Vzhledem k rozmanitosti hry šachy obsahuje protokol také příkaz pro nastavení požadované varianty hry. Dále obsahuje příkazy pro různé omezování umělých inteligencí, tedy nastavení maximálního času přemýšlení, omezení použité paměti nebo počtu jader procesoru a také maximální hloubky prohledávání umělé inteligence. Protokol podporuje také příkazy pro úpravu hrací plochy v editačním módu a také příkazy pro krok zpět. Protokol také podporuje příkaz pro nastavení určité proměnné na zadanou hodnotu, umožňuje tedy nastavení hodnot proměnným, které nejsou nijak specifikovány přímo v protokolu.

V novější verzi protokolu, tedy od verze 2, je obsazena podpora tzv. rozšíření, kdy umělá inteligence zašle seznam proměnných, které u ní lze nastavovat.

Protokol také umožňuje povolení přemýšlení i v době hry druhého hráče a s tím spojený příkaz pro předání aktuálně vypočteného tahu. Obsahuje také příkaz pro zjištění, zda již umělá inteligence vykonala všechny příkazy, které obdržela.

Podrobnější popis protokolu je uveden v "Příloha B - Chess Engine Communication Protocol".

Čerpáno z [2].

2.3 Universal Chess Interface

Universal Chess Interface (UCI) je otevřený textový komunikační protokol pro komunikaci šachových brainů s grafickým rozhraním hry. Navrhli jej v roce 2000 Rudolf Huber a Stefan Mayer-Kahlen jako konkurenci ke staršímu používanému protokolu Chess Engine Communication Protocol. Vytvoření tohoto protokolu je výsledkem snahy o zjednodušení komunikace mezi grafickým rozhraním hry a umělou inteligencí. S použitím tohoto protokolu se také přesouvá část úkolů z brainu na grafické rozhraní hry. Příkladem může být databáze začínajících tahů, kdy tahy vybírá grafické rozhraní hry a až v momentě, kdy pro danou situaci není v databázi žádný záznam, je spuštěn brain pro výpočet dalšího tahu.

Rozšíření používání tohoto protokolu nastalo po roce 2002, kdy jej začala podporovat firma Chessbase, tvůrce šachového programu Fritz. Od té doby vzniklo více než sto šachových brainů podporujících UCI.

Tento protokol stejně jako předchozí obsahuje příkaz pro určení používaného protokolu, což plyne z nutnosti odlišit od sebe dva současně používané komunikační protokoly u jedné hry. Protokol dále obsahuje příkazy pro zjištění informací o umělé inteligenci a vzhledem k využití u komerčních umělých inteligencí také příkazy pro registraci umělé inteligence či ochranu proti kopírování.

Protokol samozřejmě obsahuje také příkazy pro zahájení nové hry a předávání tahů. U těch se předává výchozí stav hrací plochy následovaný provedenými tahy. Obsahuje také příkazy pro zahájení výpočtu a jeho co nejrychlejší ukončení. Stejně jako u předchozího protokolu zde najdeme příkaz pro zjištění, zda již umělá inteligence vykonala vše, co měla. Je zde také příkaz pro nastavení předem nedefinovaných proměnných, sloužící k nastavování různých parametrů umělé inteligence.

Zajímavostí u tohoto protokolu je příkaz, který oznámí umělé inteligenci, že hráč zahrál předpokládaný tah a tím může umělá inteligence pokračovat v již rozpracovaném výpočtu.

Podrobnější popis protokolu je uveden v "Příloha C - Universal Chess Interface".

Čerpáno z [3].

3 Tvorba umělých inteligencí

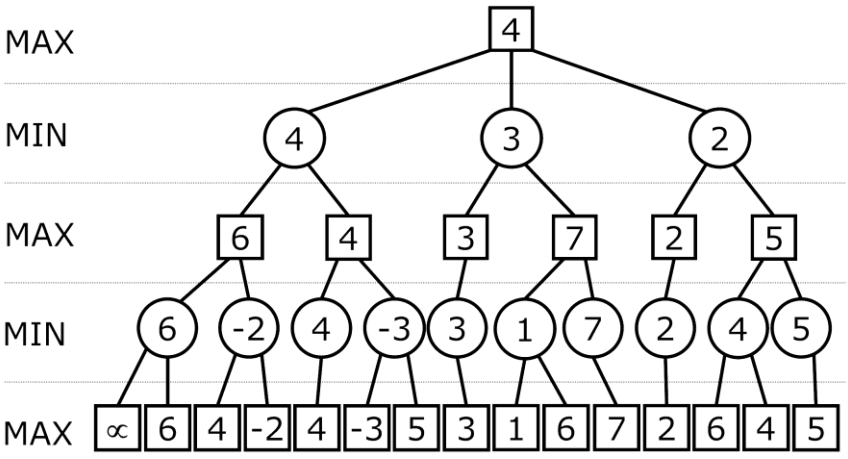
Při tvorbě umělých inteligencí pro klasické hry (šachy, dáma, …) se nejčastěji využívá prohledávání stavového prostoru. Stavový prostor je množina stavů, do kterých se může hra dostat. Jeden z těchto stavů označíme jako počáteční a některé jako koncové stavy. Koncovým stavem rozumíme stav hry, kdy hra končí, tedy kdy jeden z hráčů vyhrál nebo již není možné provést žádný další tah. Pomocí definovaných přechodů mezi jednotlivými stavy, tedy pomocí možných tahů, se snažíme najít cestu od počátečního stavu ke koncovému, který znamená náš úspěch.

Podle typu hry mohou být stavové prostory různě velké, v některých případech až nekonečné. Prohledání celých takto velkých stavových prostorů může i nejvýkonnějším počítačům zabrat roky. Proto je nutné prohledávání stavového prostoru nějakým způsobem omezit, aby se redukoval počet prohledávaných uzlů. K tomuto účelu se často používá algoritmus minimax.

3.1 Minimax

Minimax je algoritmus, který prohledává stavový prostor s omezením na určitý počet tahů. Vytvoří se stromový graf pro daný počet tahů dopředu a listy tohoto grafu jsou ohodnoceny ohodnocovací funkcí. Tato ohodnocovací funkce vrací číselné ohodnocení stavu hry, kde větší hodnota ohodnocení představuje příznivější situaci pro hráče spouštějícího algoritmus, a menší hodnota ohodnocení představuje příznivější situaci pro jeho soupeře.

První hráč se tedy snaží provést tah, který je pro něj co nejlepší, vybírá tedy větve s nejlepším ohodnocením. Druhý hráč naopak vybírá větve s co nejnižším ohodnocením. Tento výběr se provádí od listů stromu směrem ke kořeni. Název algoritmu minimax vychází právě ze střídání minima a maxima v jednotlivých úrovních stromu. S každou další prohledávanou úrovní se počet prohledávaných uzlů exponenciálně zvětšuje. U her s velkým větvícím faktorem, jako například piškvorky, zvládá algoritmus v rozumném čase prohledání jen několika málo úrovní.



Obrázek 3.1 - Algoritmus Minimax

Pseudokód:

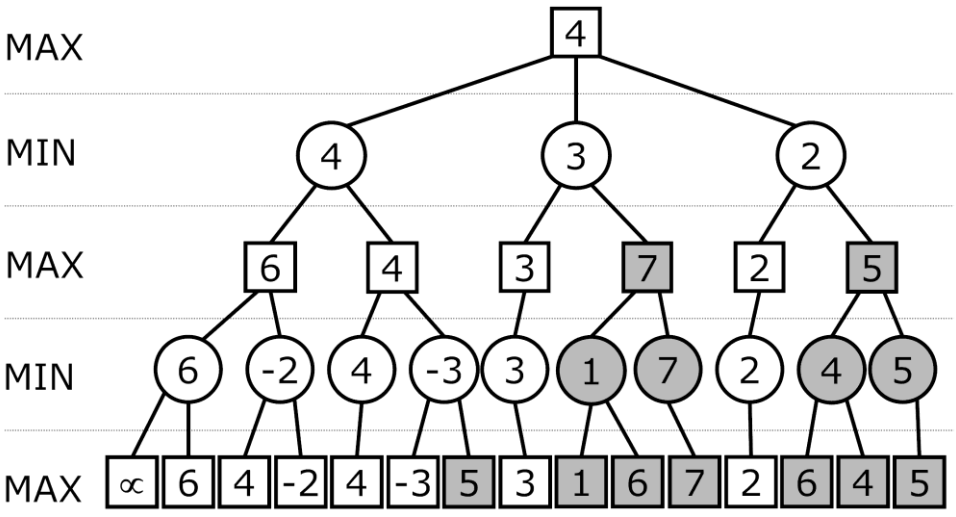
```
fun minimax(n: node, d: int): int =
   if leaf(n) or depth=0 return evaluate(n)
   if n is a max node
      v := L
      for each child of n
         v' := minimax (child,d-1)
         if v' > v, v:= v'
      return v
   if n is a min node
      v := W
      for each child of n
         v' := minimax (child,d-1)
         if v' < v, v:= v'
      return v
```

Čerpáno z [4], [5] a [6].

3.2 Alfa-beta ořezávání

Alfa-beta ořezávání je rozšíření algoritmu minimax o vynechání prohledávání větví stromu, které nemohou ovlivnit výběr nejlepšího tahu. Víme-li tedy, že je nějaký tah špatný, tedy vede k soupeřově výhře, nemá smysl dále prohledávat tuto část stromu. Tímto způsobem jsou pro výpočet nepotřebné větve stromu (v obrázku šedě) vynechány a výpočet je tím urychlen.

Účinnost tohoto vylepšení je závislá na pořadí prohledávání jednotlivých větví. Čím dříve jsou prohledány větve s nejlepším ohodnocením, tím více větví je vynecháno a tím urychlen výpočet.



Obrázek 3.2 - Algoritmus Minimax s Alfa-Beta ořezáváním

Pseudokód:

```
fun alfabeta(n: node, d: int, min: int, max: int): int =
   if leaf(n) or depth=0 return evaluate(n)
   if n is a max node
      v := min
      for each child of n
         v' := alfabeta (child,d-1,v,max)
         if v' > v, v:= v'
         if v > max return max
      return v
   if n is a min node
      v := max
      for each child of n
         v' := alfabeta (child,d-1,min,v)
         if v' < v, v:= v'
         if v < min return min
      return v
```

Čerpáno z [4], [5] a [6].

3.3 Selektivní metoda

Selektivní metoda slouží k omezení větvení stromu pro urychlení výpočtu. Principem metody vynechání bezpředmětných tahů nebo výběr pouze několika nejlepších tahů, které se budou prohledávat dále. Tím se zmenší větvení stromu, a tím i časová náročnost algoritmu.

Nevýhodou je nutnost ohodnocení jednotlivých uzlů již při větvení stromu, kvůli výběru tahů, jež se budou dále prohledávat. Na rozdíl od předchozího vylepšení, tedy Alfa-Beta ořezávání, nezaručuje selektivní metoda stejné výsledky, jako algoritmus minimax. Může tedy nastat situace, kdy se vynecháním uzlu, který je v daný okamžik méně výhodný, můžeme vyhnout prohledání uzlu, jež by vedl k lepšímu celkovému ohodnocení.

4 Návrh aplikace

V této kapitole se podíváme na návrh jednotlivých částí našeho nástroje.

Při návrhu této aplikace byl kladen velký důraz na univerzálnost. Hry, které je možné v této aplikaci hrát, nejsou přímo součástí aplikace, ale jsou načítány z dynamických knihoven. To umožní dodatečné přidání další hry, bez nutnosti kompilace celé aplikace. Pro uživatele to znamená stažení pouze příslušné knihovny, pro přidání další hry.

4.1 Návrh vlastního komunikačního protokolu

V kapitole 2 - Používané komunikační protokoly jsme si mohli všimnout společných rysů uvedených protokolů, i když se jednalo o protokoly pro různé hry. V této kapitole se pokusím z těchto společných rysů pro různé etapy hry navrhnout vlastní univerzální komunikační protokol mezi umělou inteligencí a grafickým rozhraním, který bude použitelný jak pro zmíněné hry, tak pro další podobné hry.

4.1.1 Společné fáze u různých protokolů

U zmíněných protokolů jsme si mohli všimnout, že použití externí umělé inteligence prochází třemi základními fázemi - inicializací brainu, nastavením hry a průběhem samotné hry.

Ve fázi inicializace brainu probíhá nastavení komunikačního protokolu, který bude použit, a jeho verze - většinou nejnovější podporované oběma stranami. Dále se zjišťují informace o připojené umělé inteligenci, tedy její název a údaje o autorovi, případně další podrobnější informace. Také se zde může objevit ověření oprávnění použití umělé inteligence, například její registrací.

V další fázi probíhá nastavení hry před jejím spuštěním. Může jít jak o rozměry hracího pole, tak o různé varianty hry. Nastavuje se také omezení dostupných prostředků pro potřeby brainu, tedy počet jader procesoru, množství paměti a čas pro výpočet tahu. Tyto parametry se nastavují většinou ještě před spuštěním hry, je však možné je v průběhu hry měnit.

Poslední fází je průběh samotné hry. V této fázi se předávají mezi grafickým rozhraním a umělou inteligencí uskutečněné tahy jednotlivých hráčů a příkazy pro začátek a konec přemýšlení, případně pro zaslání aktuálně nejlepšího nalezeného tahu při kontinuálním přemýšlení.

Některé příkazy se nedají zařadit do žádné z uvedených fází. Jde například o zasílání chyb či ladících informací, nebo zpráv pro kontrolu připravenosti brainu zpracovávat další příkazy.

4.1.2 Předpoklady univerzálního protokolu

Vzhledem k tomu, že náš protokol má být univerzální, a tedy použitelný ve více typech her, musíme při jeho návrhu přistoupit k určitým kompromisům. Protokol musí pokrývat takové příkazy, které

jsou potřebné u všech typů her. Příkazy specifické pro konkrétní hry, vzhledem k jejich množství, pokrývat nemůže, musí však být schopný je předávat i bez jejich znalosti. Musí také poskytovat možnost informování o tom, že daný příkaz je druhou stranou podporován.

Použití protokolu u více typů her znamená nutnost důkladnější identifikace umělé inteligence ve fázi jeho inicializace. Nestačí pouze ověřit název protokolu, jako tomu je u jednoúčelových protokolů. Kromě názvu protokolu je nutné také ověřit, zda se jedná o stejný typ hry. I když je protokol navržen s ohledem na univerzálnost a rozšiřitelnost, je vhodné předpokládat, že se při pozdějším použití protokolu u nových her objeví potřeba protokol upravit či přidat nové příkazy. Z toho důvodu je potřeba kontrolovat při identifikaci umělé inteligence také verzi použitého protokolu.

4.1.3 Universal Brain Protocol

Nyní si představíme námi navržený komunikační protokol. Vzhledem k jeho účelu jsme jej pojmenovali Universal Brain Protocol (UBP). Jde o textový protokol navržený pro komunikaci grafické části hry a umělých inteligencí pro různé typy her. Používá jednoslovné příkazy s parametry definovanými dle daného příkazu. Na každý příkaz je druhou stranou zaslána odpověď s požadovanou informací, potvrzením příkazu či informací o chybě. Výjimku tvoří příkazy zasílající chyby, zprávy uživateli nebo ladící informace, a příkazy pro kontrolu připravenosti brainu a ukončení brainu.

Následující podkapitoly obsahují seznam příkazů s jejich popisem, utříděné podle jejich použití.

Pro zápis částí komunikace budeme používat následující formát:

G: PŘÍKAZ <parametr>

B: PŘÍKAZ <parametr>

kde "G:" značí, že příkaz je posílán grafickou částí hry brainu, a "B:", že je příkaz posílán brainem. Příkazy mohou být také posílány rychleji, než je druhá strana schopna odpovídat. Odpovědi pak nemusí následovat okamžitě za příkazem, ale mohou být zasílány v pořadí příchodů příkazů, nebo v pořadí jiném - podle charakteru příkazu a implementace brainu. Brain by měl být navíc schopný přijímat a reagovat na příkazy, i když právě vykonává jinou časově náročnou činnost, například výpočet dalšího tahu.

Inicializace brainu

Nastavení protokolu

G: UBP

B: UBP OK

Po spuštění brain očekává příkaz pro nastavení použitého komunikačního protokolu. Grafické rozhraní hry pošle příkaz UBP (Universal Brain Protocol) pro nastavení brainu na tento protokol. Je-li

brain schopen tímto protokolem komunikovat, odpoví UBP OK. Nepřijde-li grafickému rozhraní tato odpověď, ví, že brain tento protokol nepodporuje, a může vyzkoušet jiný komunikační protokol nebo brain ukončit.

Nastavení verze protokolu

G: UBP_VERSION <int>

B: UBP_VERSION [OK | ERR]

Pro správné používání komunikačního protokolu je nutné dohodnout se na verzi protokolu, která bude používána. Grafické rozhraní zašle příkaz **UBP_VERSION** a jako parametr uvede číslo verze, kterou chce používat. Podporuje-li brain tuto verzi protokolu, potvrdí její použití příkazem **UBP_VERSION OK**, jinak odpoví **UBP_VERSION ERR**. Obdrží-li grafické rozhraní od brainu negativní odpověď, opakuje zasílání tohoto příkazu s nižším číslem verze protokolu, pokud jej podporuje.

Nastavení hry

G: GAME "<name>"

B: GAME [OK | ERR]

Poté, co je dohodnuta použitá verze protokolu, je nutné nastavit typ hry, pro kterou má být protokol použit. Tento příkaz neslouží ani tak jako informace pro brain, kterou hru použít, jako spíš ověření, že grafickým rozhraním byl vybrán správný brain. Grafické rozhraní zasílá brainu příkaz **GAME,** u kterého jako parametr uvede v uvozovkách typ hry, která má být hrána, například chess či gomoku. Podporuje-li brain tento typ hry, odpoví **GAME OK,** jinak **GAME ERR**. V případě, že grafické rozhraní dostane negativní odpověď od brainu, nezbývá mu většinou, než brain ukončit, jelikož není vhodný pro hru implementovanou grafickým rozhraním.

Identifikace brainu

G: ABOUT

B: ABOUT name="<name>"; author="<author>"; [email="<email>";]
[address="<address>";] [phone="<phone>";] [date="<date>";] […]

Pro získání informací o použitém brainu zašle grafické rozhraní příkaz **ABOUT**, na který brain odpoví stejným příkazem, kde jako parametry bude mít uvedeny informace, které o sobě chce uvést. Povinnými informacemi jsou název brainu a jméno autora. Nepovinné informace mohou být e-mail autora, jeho adresa, telefon, či datum vytvoření brainu. Díky zápisu informací jako dvojice název="hodnota" je možné uvést i další informace, avšak není zaručeno, že jim bude grafické rozhraní rozumět.

Zjištění dostupných nastavení brainu

G: GET_OPTIONS

B: GET_OPTIONS <name>="hodnota"[;<name>="hodnota"]

Poslední fází inicializace brainu je zjištění dostupných nastavení brainu a jejich defaultních hodnot. Tyto nastavení může mít každý brain jiné, proto jsou po spuštění brainu zjištěny dostupné možnosti nastavení a podle toho je v grafické části hry zobrazeno příslušné nastavení pro daný brain.

Nastavení hry

Zaslání příkazu

G: DO <akce>

B: DO OK [<odpověď podle akce>]

B: DO ERR ["<popis>"]

Příkazem **DO** můžeme přinutit brain provést nějakou činnost, která je typická pro daný typ hry. Dojde-li k úspěšnému provedení akce, odpoví brain zprávou **DO OK**, za kterou případně připojí výsledek akce. V případě chyby informuje zprávou **DO ERR** a případným podrobnějším popisem chyby.

Nastavení proměnných podle hry

G: SET <name>="<value>"[; <name>="<value>"]

B: SET OK

B: SET ERR <name>[="<popis>"][;<name>[="popis"]]

Pomocí příkazu **SET** můžeme nastavit různé proměnné. Může jít například o typické nastavené pro danou hru (například rozměry hrací desky u piškvorek), nebo o konkrétní nastavení brainu, které zveřejnil příkazem **GET_OPTIONS**. Jedním příkazem lze nastavit současně více proměnných. V případě chyby některé z nich je zaslána zpráva **SET ERR** s dvojicemi názvu proměnné a příslušné chyby (například očekávání čísla místo textu). Je-li vše bez chyb, dojde k potvrzení zprávou **SET OK**.

Zjištění hodnoty proměnné

G: GET <name>[;<name>]

B: GET <name>="<value>"[;<name>="<value>"]

Potřebuje-li grafická část hry zjistit hodnotu nějaké proměnné, požádá o ni příkazem **GET**, za kterým následuje výčet proměnných, jejichž hodnoty požaduje. Brain odpoví opět příkazem **GET**, za kterým následuje seznam názvů proměnných a jejich hodnot.

Nová hra

G: NEW_GAME

B: NEW_GAME OK

Tímto příkazem dojde k vytvoření nové hrací plochy s počátečním stavem hry.

Načtení rozehrané hry

G: LOAD <pos1>=<val1>;[<pos2>]=<val2>];…

B: LOAD [OK | ERR] ["<popis_chyby>"]

Načtení rozehrané hry můžeme provést pomocí příkazu **LOAD**, za kterým uvedeme seznam všech obsazených polí a jejich hodnot. V případě chyby odpoví brain zprávou **LOAD ERR** a případným popisem chyby. Je-li vše v pořádku, dojde k potvrzení příkazem **LOAD OK**. Před samotným naplněním hrací desky uvedenými hodnotami dojde k jejímu smazání, není tedy nutné uvádět příkaz **NEW_GAME**.

Průběh hry

Táhnutí

G: TURN <tah>

B: TURN OK

B: INVALID TURN

Chce-li grafická část hry informovat brain o provedeném tahu druhého hráče, použije k tomu příkaz **TURN**. Za tímto příkazem následuje tah, který protivník provedl. Zápis tahu se liší pro každý typ hry. Například pro piškvorky má tah formu souřadnic tahu, tedy [x],[y]. Pro dámu bude mít tah tvar například b2d4f6, což značí dvojskok z pole B2, na D4 a následně na F6. V případě chybně zadaného tahu (např. mimo pole nebo na obsazené pole), odpoví brain příkazem **INVALID_TURN.** Je-li tah v pořádku, je potvrzen zprávou **TURN OK**. Po samotném předání tahu však brain nezačne počítat další tah, čeká až na povel od grafické části hry.

Tah brainu

G: GO

B: TURN <tah>

Povel k započetí výpočtu dalšího tahu je předán příkazem **GO**. Poté, co brain vypočte příslušný tah, zašle jej grafické části pomocí příkazu **TURN** následovaným příslušným zápisem tahu pro danou hru.

Krok zpět

G: TAKEBACK <x>,<y>=<h>;[<x>,<y>=<h>;]…

B: TAKEBACK OK

Požaduje-li grafická část hry po brainu krok zpět, je zaslán příkaz **TAKEBACK**, za kterým následují souřadnice změněných polí a hodnoty, na které se tato pole mají změnit. O uchovávání historie tahů se tedy stará grafická část hry. Při kroku zpět se vrací až za poslední tah lidského hráče, aby měl krok zpět smysl. V rámci jednoho příkazu jsou tedy předány až dva tahy a v každém tahu může být změněno několik polí. Potvrzení úspěšného provedení kroku zpět je provedeno příkazem TAKEBACK OK.

Kontrola připravenosti brainu

G: READY

B: READY OK

Potřebuje-li grafická část hry zjistit, zda už je brain připraven vykonávat další činnost, zašle mu příkaz **READY** a čeká na odpověď. V okamžiku přijetí odpovědi **READY OK** má potvrzeno, že brain vykonal veškeré výpočty a čeká na další příkazy.

Ukončení brainu

G: END

Chceme-li brain ukončit, zašleme mu příkaz END. Na tento příkaz by měl brain co nejrychleji zareagovat bez zaslání další zprávy se ukončit. Nedojde-li k ukončení do určitého časového limitu, je možné, že bude brain násilně ukončen.

Příkazy zasílané brainem

Neznámý příkaz

B: UNKNOWN (<příkaz>)

Příkazem UNKNOWN informuje brain grafickou část hry, že dostal příkaz, který nezná. V závorce je uveden příkaz, o který se jedná.

Chyba

B: ERROR "<msg>"

Vznikne-li při provádění nějaké akce v brainu chyba, informuje o tom příkazem **ERROR** grafickou část hry. Uživatel by měl být informován o výskytu chyby a měla by mu být poskytnuta informace o chybě obsažená v parametru msg za tímto příkazem.

Zpráva

B: MESSAGE "<msg>"

Chce-li brain informovat uživatele o nějaké události či stavu, může k tomu využít příkaz MESSAGE. Za tento příkaz připojí zprávu pro uživatele.

Ladící informace

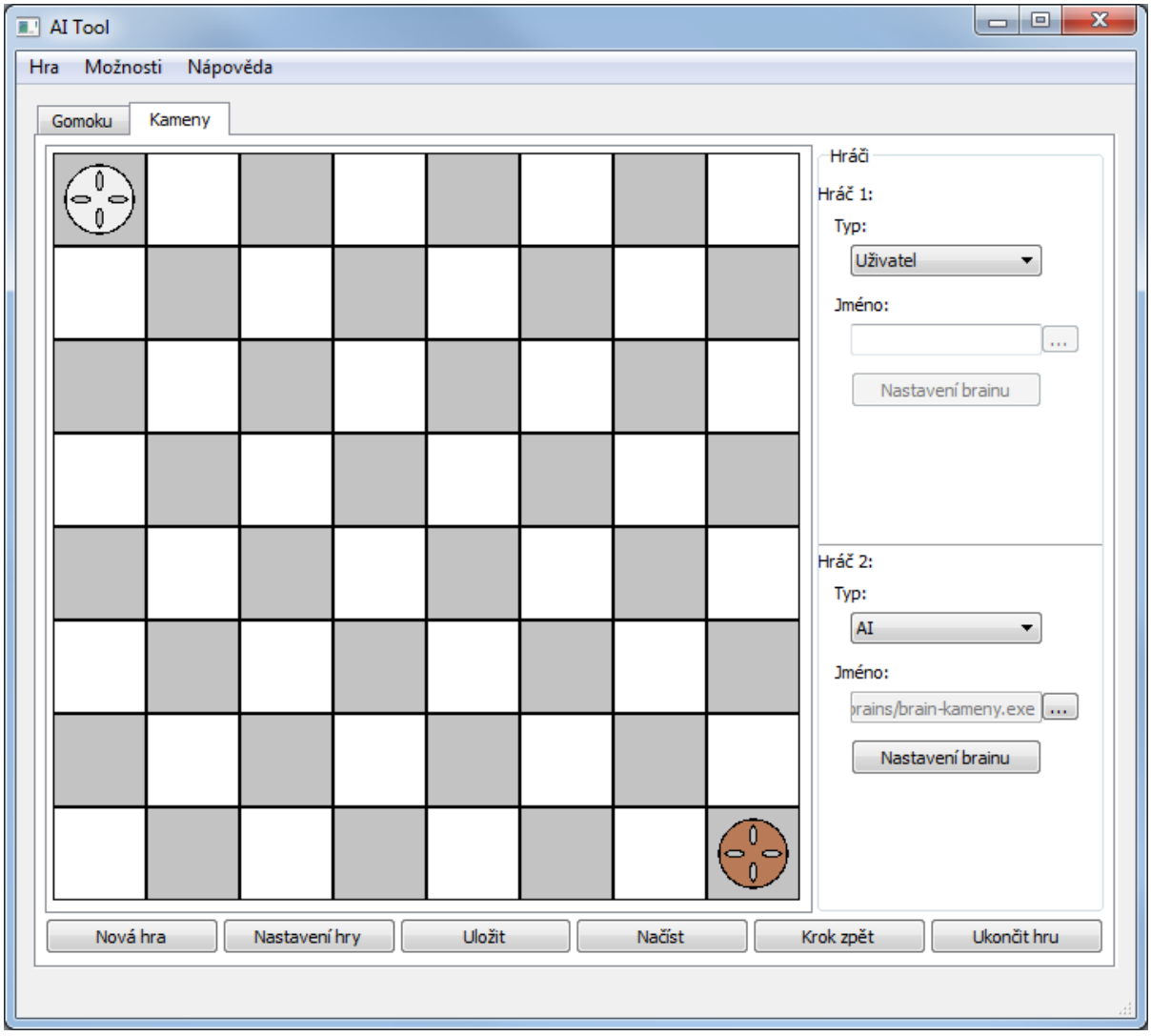
B: DEBUG <JSON>

Je-li proměnná debugEnabled nastavena na 1, jsou po zaslání vybraného tahu brainem zaslány také ladící informace, tedy ohodnocení jednotlivých polí. Tyto informace jsou předány ve formátu JSON, který umožní jednoduché zpracování zaslaných dat. Formát dat uložených v JSON je {"dbg":[]}, kde uvnitř pole jsou další pole: [<x>,<y>,<h>,<d>]. <x>,<y> a <h> jsou číselná hodnoty souřadnic a ohodnocení daného pole. <d> je další pole obsahující opět záznamy typu [<x>,<y>,<h>,<d>], jsou v něm tedy uloženy ohodnocení polí po zanoření v příslušném poli.

4.2 Grafické uživatelské rozhraní

4.2.1 Vzhled aplikace

Uživatelské rozhraní je jednoduché a přizpůsobené hlavnímu účelu aplikace, tedy výběr a spuštění požadované hry s možností výběru mezi lidskými hráči a umělou inteligencí. V podstatě celou plochu aplikace zabírají záložky s jednotlivými hrami. Hlavním prvkem je zde hrací deska, která je zprava a zespodu obklopena dalšími ovládacími prvky. Vpravo od hrací desky jsou nastavení týkající se jednotlivých hráčů, tedy výběr mezi lidským hráčem a umělou inteligencí, výběr umělé inteligence a přístup k nastavení umělé inteligence. Pod hrací deskou jsou umístěna tlačítka pro přístup k akcím spojených s hrou, jako jsou start nové hry, nastavení hry, uložení a načtení stavu hry, krok zpět a ukončení hry.



Obrázek 4.1 - Grafické rozhranní aplikace

4.2.2 Menu

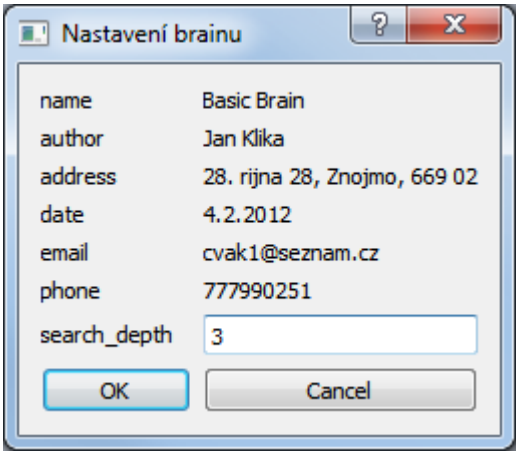
Možnosti spuštění nové hry a uložení a načtení stavu hry jsou dostupné také z menu aplikace pod nabídkou Hra. Kromě zmíněných zde najdeme také zobrazení pravidel aktuálně vybrané hry a ukončení aplikace.

Pod nabídkou Možnosti v menu aplikace najdeme možnost zapnutí či vypnutí zobrazení ladících informací, tedy ohodnocení jednotlivých polí. Pro zobrazení těchto informací však musí být alespoň jedním z hráčů umělá inteligence připojená protokolem UBP, která tyto informace poskytuje. Dále zde najdeme možnost zapnutí krokování tahů pro zobrazování tahů umělé inteligence až po potvrzení uživatelem. Je zde také možnost nastavit minimální čas tahů umělé inteligence, sloužící pro zpomalení hry a zpřehlednění jednotlivých tahů, avšak bez nutnosti každý tah potvrdit. Poslední možností tohoto menu je nastavení jazyka aplikace. Hlavním jazykem aplikace je čeština. Tato možnost je dostupná, pouze je-li dostupná také anglická lokalizace.

Poslední nabídkou v menu je Nápověda. Zde najdeme zobrazení nápovědy k používání této aplikace, zobrazení informací o použité verzi Qt a informace o verzi a autoru aplikace.

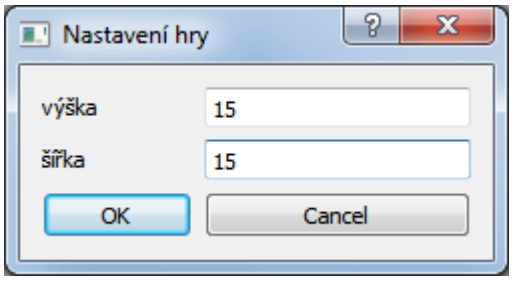
4.2.3 Nastavení

Po vybrání umělé inteligence jsou načteny informace o ní a autorovi, a také její dostupné nastavení. Toto nastavení se pak zobrazí při kliknutí na tlačítko Nastavení brainu. U každé umělé inteligence používající protokol UBP se tedy zobrazují pouze ty nastavení, jež jsou u ní dostupné.



Obrázek 4.2 - Nastavení brainu

Dostupná nastavení pro každou hru jsou získávána z příslušné knihovny dané hry. Tato nastavení jsou pak zobrazena kliknutím na Nastavení hry.



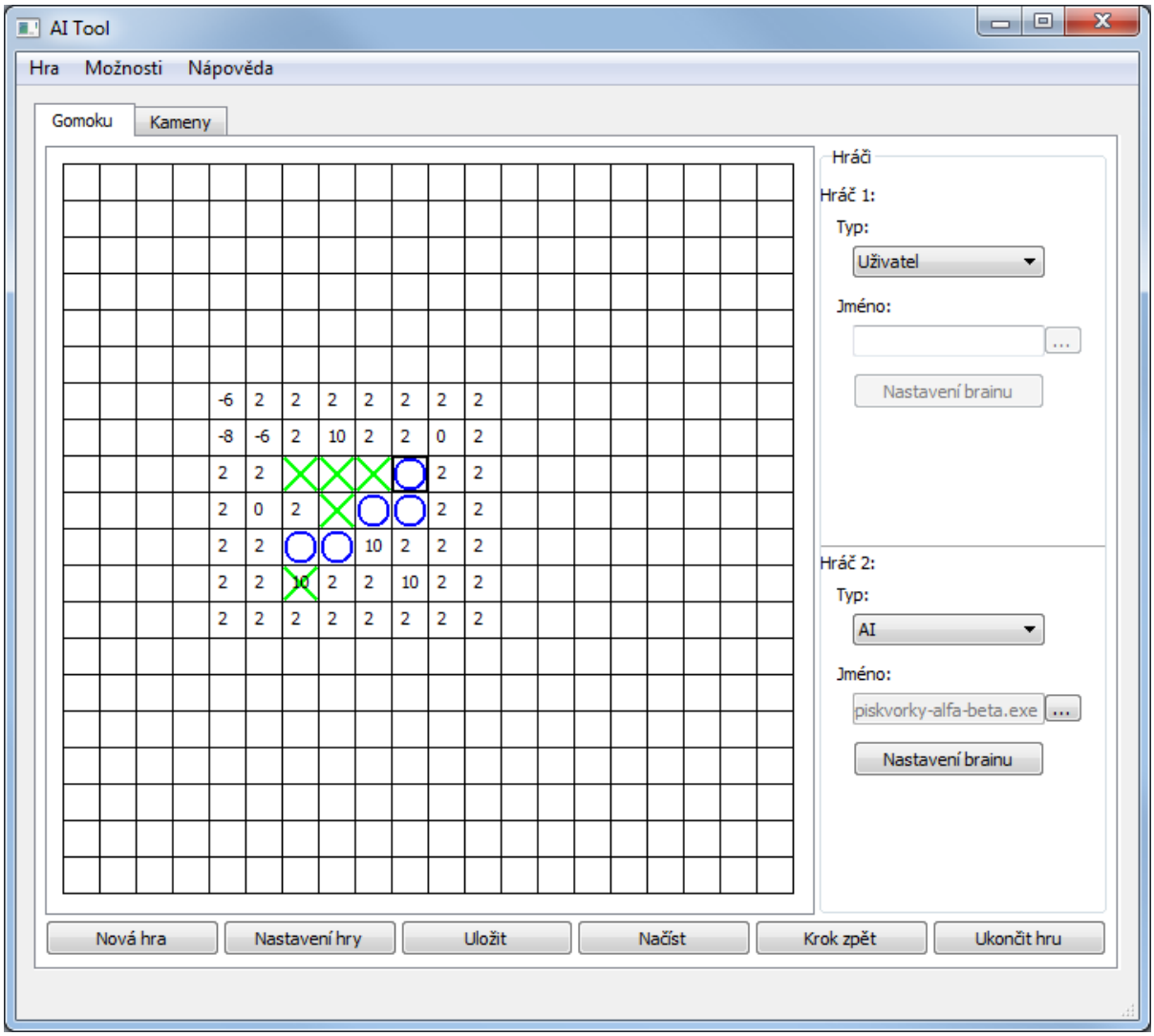
Obrázek 4.3 - Nastavení hry

4.2.4 Ladící informace

Účastní-li se hry alespoň jedna umělá inteligence používající protokol UBP, která umožňuje předávání ladících informací, jsou tyto ladící informace zobrazeny po povolení Debug z menu Možnosti. Tyto ladící informace jsou zobrazovány jako čísla v příslušných políčkách hrací desky, která reprezentují ohodnocení těchto políček. Zobrazovat lze i více zanoření ohodnocení. K zobrazení dalšího zanoření dojde kliknutím pravým tlačítkem myší na ohodnocené políčko, čímž se zobrazí ohodnocení ostatních políček, která přispěla k ohodnocení vybraného políčka. Kliknutím pravým tlačítkem myši na políčko bez ohodnocení dojde k zobrazení základní úrovně ladících informací.

Při zapnuté možnosti krokování tahů se ohodnocení jednotlivých polí zobrazí ihned po vypočtení tahu umělou inteligencí. Samotný vypočtený tah se však zobrazí až po potvrzení daného tahu.

Hodnoty, význam i počet zanoření závisí čistě na umělé inteligenci, která požadované informace předává ve formátu definovaném v definici protokolu UBP (viz str. 18 - Ladící informace), tedy JSON.



Obrázek 4.4 - Zobrazení ladících informací

4.3 Samostatné hry

Jak již bylo zmíněno, kvůli rozšiřitelnosti jsou samostatné hry odděleny od aplikace a jsou umístěny v dynamických knihovnách. Ty se automaticky načtou po spuštění aplikace. Tyto knihovny se musí starat o vše týkající se dané hry, tedy o vykreslování hrací desky, střídání hráčů, kontrolu platnosti tahů a výhry, a také o načítání a komunikaci s umělými inteligencemi.

Obsažené hry jsou navrženy obecněji, než by bylo nutné. Byla snaha navrhnout je tak, aby se daly snadno předělat na jinou hru. Například předávání tahu není omezeno počtem předaných souřadnic, což umožní předání tahu pro piškvorky, kdy jsou tahem pouze jedny souřadnice, stejně tak předání tahu pro hru dáma, kdy může být tahem několik souřadnic značících braní několika figur.

Hry jsou také připraveny pro použití více komunikačních protokolů, což umožní připojení již existující umělé inteligence pomocí již používaného protokolu a následně například sehrání zápasu mezi touto umělou inteligencí a svojí vlastní, pro jejich porovnání.

Dynamické knihovny s hrami jsou tedy navrženy tak, aby pouhou změnou vykreslování hrací desky a logiky hry došlo k rychlému vytvoření další hry, která bude aplikací podporována.

4.4 Umělé inteligence

Součástí práce jsou také dvě testovací umělé inteligence pro hry piškvorky a kameny, a také kostra umělé inteligence. Jde vlastně o obálku umělé inteligence, která obsahuje příkazy protokolu UBP. Do té pak stačí vložit samotnou logiku hry a napojit příkazy protokolu na příslušné akce. Tato kostra je navržena tak, aby umožnila vyhnout se opakovanému vytváření stejného kódu pro realizaci komunikace s grafickou částí hry, a tím umožnila soustředit se na vývoj samotné umělé inteligence.

Umělé inteligence, které jsou součástí této práce, jsou navrženy jako demonstrace použití komunikačního protokolu UBP a možnosti zobrazení ladících informací v našem nástroji. V žádném případě si nekladou za cíl hrát bezchybnou a rychlou hru. Slouží hlavně jako ukázka jedné z mnoha možností implementace umělé inteligence pro náš nástroj. Ostatně vývoj rychlých a silných umělých inteligencí je prací uživatelů tohoto nástroje, jež má být použitím tohoto nástroje usnadněna.

5 Implementace

V této části si popíšeme způsob implementace důležitých částí našeho nástroje. Nejprve se zaměříme na popis hlavní aplikace. Dále se seznámíme s implementací her připojitelných do našeho nástroje, a nakonec s implementací umělých inteligencí pro protokol UBP.

Celá tato práce je implementována v jazyce C++ a frameworku Nokia Qt4. Tento framework umožňuje snadné vytváření multiplatformních grafických aplikací a usnadňuje programování již implementovanými třídami řešícími časté problémy. Obrovskou výhodou tohoto frameworku je také velmi obsáhlá dokumentace se spoustou příkladů.

5.1 Nástroj

5.1.1 Třída hlavního okna

Hlavní třídou nástroje je třída `MainWindow`. Ta se stará o zobrazení hlavního okna aplikace, informací na stavovém řádku a pop-up oken s různými informacemi od hry. Nejdůležitější funkcí této třídy je funkce `loadPlugins` spouštěná při spuštění aplikace, která se stará o načtení dynamických knihoven s hrami ze složky `plugins`. To probíhá pomocí nástroje `PluginLoader`, který načítá jednotlivé soubory z dané složky a pokouší se je přetypovat na objekt typu `pluginInterface`. Dojde-li k úspěšnému přetypování, víme, že byla načtena knihovna s hrou, a je volána funkce `loadPlugin`. Tato funkce vytvoří nový objekt typu `Tab`, předá mu načtený plugin, propojí signály pro zobrazování pop-up oken a přidá tento objekt do panelu záložek v hlavním okně.

5.1.2 Rozhraní

Objekt typu `pluginInterface` je instance třídy splňující rozhraní pro hru připojitelnou k našemu nástroji. Definice rozhraní obsahuje funkce pro předání ukazatele na kreslící plátno, zjištění jména hry, nastavení hráče, začátek nové hry, zjištění, zda právě probíhá hra, zjištění jména a typu hráče, ukončení hry, získání a nastavení možností hry a umělé inteligence, získání informací, které o sobě zveřejnil brain, provedení kroku zpět ve hře, uložení a načtení stavu hry. Kromě těchto funkcí týkajících se průběhu hry obsahuje také funkce pro získání pravidel daní hry, předání ukazatele na ladící informace, povolení či zakázání zobrazování ladících informací a nastavení možností krokování a zpomalení tahu umělé inteligence.

Dále jsou definovány sloty `onClicked` a `onKeyPressed` pro předání události kliknutí myší a stisku klávesy a `paint` pro předání události překreslení hrací desky.

Poslední částí definice rozhraní her je definice signálů. Zde jsou definovány signály pro předání běžné zprávy, chybové zprávy a informace o výhře. Dále je zde definován signál `sDebug`, pomocí kterého se předávají ladící informace z brainu.

5.1.3 Záložka se hrou

Další důležitou třídou nástroje je třída `Tab`. Ta dědí od třídy `QWidget` a slouží jako záložka s hrou v aplikaci. Jak již bylo zmíněno, tato třída je vytvářena pro každý načtený plugin se hrou, tedy pro každý objekt typu `pluginInterface`. Hlavní náplní třídy je zobrazení ovládacích prvků v záložce hry (viz Vzhled aplikace) a komunikace s načtenou knihovnou hry pomocí definovaného rozhraní `pluginInterface`.

Konstruktoru třídy je předán ukazatel na příslušný objekt typu `pluginInterface` spolu s názvem souboru, ze kterého byl objekt načten. V tomto konstruktoru je mimo jiné předán ukazatel na kreslící plátno do knihovny s hrou, což umožňuje snadné vykreslování hrací desky přímo z knihovny dané hry. Dále je nastaveno zachytávání stisku kláves a tlačítek myši na hrací desku a příslušné signály jsou předávány objektu rozhraní.

Tato třída zprostředkovává komunikaci mezi uživatelem a danou hrou. Pomocí tlačítek předává knihovně hry příkazy typu start hry, konec hry, krok zpět a uložení a načtení hry. Při změně hráče je hra ukončena a informace o hráči předány knihovně hry. Jde-li o umělou inteligenci, je předána cesta k aplikaci umělé inteligence, která byla získána pomocí dialogového okna pro výběr souboru. Knihovna hry si ihned ověří, zda jde o platnou umělou inteligenci pro danou hru a v případě chyby je uživatel informován chybovým hlášením. Uložení a načtení hry je realizováno stejně jako výběr umělé inteligence, je tedy předána pouze cesta k souboru a o vše ostatní se již stará příslušná knihovna dané hry.

Další důležitou funkcí této třídy je zobrazení nastavení hry a umělých inteligencí. Zobrazené oba druhy nastavení fungují na stejném principu. Od knihovny hry je vyžádáno asociativní pole, kde klíčem je název parametru a hodnotou je jeho aktuální hodnota. Tyto dvojice jsou pak zobrazeny v dialogovém okně ve formě formuláře (viz 4.2.3 - Nastavení). Při zobrazení nastavení umělé inteligence jsou před samotnými nastavitelnými hodnotami zobrazeny také informace, které o sobě umělá inteligence poskytuje, jako například název brainu, jméno a kontakt na autora a další.

Během hry samotné se tato třída vůbec nezajímá o stav hry, pouze předává události kliknutí myší na kreslící plátno nebo stisky kláves. O zpracování těchto událostní a příslušné reakce na ně se stará knihovna dané hry.

Při změně aktivní záložky se hrou dojde také ke změně zatržení možností Debug a Krokování v menu podle nastavení aktuálně vybrané hry.

5.1.4 Ladící informace

Poslední důležitou třídou nástroje je třída `Debug`. Tato třída uchovává ladící informace poskytované umělou inteligencí a poskytuje je knihovně hry pro případné vykreslení.

Data jsou této třídě předávána pomocí typu `QVariant`, což je vestavěný typ frameworku Qt, který funguje na principu unie. V tomto typu jsou předávána kvůli snadnému načtení dat ve formátu JSON právě do tohoto typu `QVariant`. Tato data jsou následně převedena do vlastního formátu uložení dat, tedy asociativního pole, kde klíčem jsou souřadnice pole, jehož se ohodnocení týká, a hodnotou je struktura obsahující hodnotu samotného ohodnocení a další asociativní pole, stejné jako předchozí klíč, které slouží pro uchování ohodnocení další úrovně.

Struktura uložení ladících informaci:

```
struct DbgRecord {
    int hodnota;
    QMap<QPair<int,int>,DbgRecord> dalsi;
};
typedef QMap<QPair<int,int>,DbgRecord> t_dalsi;
```

Při vykreslování ladících informací se pak získává ukazatel na příslušné asociativní pole. Používání ukazatelů umožňuje zanořování do různých úrovní ohodnocení. Kliknutím na ohodnocené hrací pole s žádostí o zanoření je předán funkci pro zanoření také ukazatel, který je aktuálně používán, a podle toho je zjištěno, zda se dá zanořit do další úrovně. Dá-li se zanořit, je předán ukazatel na příslušné asociativní pole s příslušným zanořením. V případě, že se zanořit hlouběji nedá, je vrácena hodnota NULL, načež je o tomto uživatel informován.

Je-li žádost o zanoření provedena nad polem, které není ohodnocené, je vrácen ukazatel na ohodnocení nejvyšší úrovně.

5.2 Knihovna hry

Zatímco v předchozí kapitole jsme se zabývali jakousi obálkou pro zobrazení jednotlivých her, nyní se již podíváme na implementaci knihoven jednotlivých her, ve kterých jde čistě o funkcionalitu dané hry.

5.2.1 Hlavní třída hry

Třída `Plugin` je hlavní třídou dynamické knihovny se hrou. Tato třída implementuje rozhraní `pluginInterface` a slouží pro zaobalení jednotlivých tříd v knihově hry a komunikaci mezi nimi. Hlavní funkcí této třidy je uchovávání stavu hry, tedy určovat, který hráč může právě hrát, a provádění příkazů měnících stav hry, jako je start hry a konec hry a uložení a načtení herní plochy.

Uchování informace o hráči, jenž je na řadě, je prováděno pomocí proměnné `actPlayer`. Není-li aktivní hra, je tato proměnná rovna nule, jinak obsahuje číslo hráče, od kterého je očekáváno zadání tahu. Toto číslo hráče se mění po úspěšném provedení tahu.

Uložení hrací plochy probíhá vytvořením cílového souboru, jehož cestu jsme získali od aplikace. Do něj se uloží hráč, jenž je na tahu, případně další potřebné informace jako rozměry herní plochy, a dále je od třídy hrací plochy vyžádán seznam obsazených polí hrací plochy, které jsou následně také uloženy do souboru. Soubor s uloženou hrou má standardně příponu `.sav`, uložené informace na začátku souboru jsou odděleny středníkem a následný seznam obsazených polí má formát `x,y=h;`, kde `x`, `y` jsou souřadnice pole a `h` je hodnota tohoto pole, tedy například číslo hráče či figury. Vzhledem k tomu, že se o uložení stará knihovna každé hry sama, není problém ukládat hry v jiném formátu.

Načtení uložené hry probíhá podobně jako její uložení. Po načtení dat do třídy hrací plochy je však nutné předat stav hrací plochy také hráčům umělých inteligencí. K tomu se použije třída `COMM`, jež slouží pro předávání povelů umělým inteligencím. Této třídě předáme ve funkci `load` načtený stav herní plochy a číslo hráče, jenž je na tahu. Třída `COMM` se pak postará o předání požadovaných informací hráčům, za které hraje umělá inteligence.

Při startu nové hry se vytvoří nová hrací plocha, tedy nový objekt třídy `Deska`. Dále se zajistí běh umělých inteligencí pomocí funkce `assureRun` třídy `COMM`. Toto je prováděno kvůli možnému automatickému ukončení některých umělých inteligencí po konci hry, tedy po výhře či prohře. Dále se umělým inteligencím zašle nastavení hry a nastavení těchto hráčů. Nastavení hry jsou ukládána v třídě `Settings` a nastavení jednotlivých hráčů v třídě `Player`, přes třídu `Settings` přístupnou. Nakonec je nastaven jako aktivní hráč číslo 1, což znamená případné zaslání příkazu pro zahájení hry začínajícímu hráči, jde-li o umělou inteligenci.

Při změně jednoho z hráčů je ukončena případná spuštěná umělá inteligence. Je-li nově nastavený hráč umělá inteligence, dojde k jejímu spuštění. V případě chyby se zachová stávající nastavení hráčů. Je-li umělá inteligence úspěšně spuštěna, dojde k načtení jejího nastavení a informací o této inteligenci. Je-li hráčem lidský hráč, je do nastavení tohoto hráče uloženo jeho jméno a typ.

Tato třída se stará také o krokování tahů umělých inteligencí a jejich zpožďování. Je-li zapnutá možnost krokování tahů, je tah přijatý od umělé inteligence uložen a označen jako aktivní. Po obdržení signálu stisknuté klávesy a ověření, že jde o klávesu mezerník, se volá funkce, jež uložený aktivní tah předá k provedení třídě hrací plochy a tah označí jako neaktivní. To zabrání jeho opakovanému provedení. Při zapnuté možnosti zpožďování tahů umělých inteligencí je při zasílání příkazu k započetí výpočtu umělé inteligence uloženo časové razítko s přesností na milisekundy. Při následném obdržení vypočteného tahu od umělé inteligence je porovnán rozdíl uloženého časového razítka od aktuálního času s nastavenou hodnotou minimální délky tahu. Trval-li výpočet tahu umělé inteligenci déle, než nastavená hodnota, je tah rovnou předán do hrací desky k provedení. Byla-li

umělá inteligence rychlejší, tah je uložen stejně, jako v případě krokování, a je nastaven čítač na hodnotu zbylého času. Po vypršení tohoto čítače je, stejně jako u krokování tahů, volána funkce pro označení tahu jako neaktivního a jeho předání do třídy hrací desky. Tento způsob navíc umožňuje, že i přes nastavenou minimální dobu tahu umělé inteligence je možné tento tah urychlit stisknutím mezerníku.

Ostatní funkce třídy slouží pro předávání informací mezi rozhraním `pluginInterface` a ostatními třídami hry a mezi ostatními třídami hry mezi sebou.

5.2.2 Nastavení hry a hráčů

Jak již bylo zmíněno v přechozí podkapitole, třída `Settings` slouží pro uchování nastavení hry. Tímto nastavením může být například šířka a výška hrací plochy u piškvorek. Dále tato třída obsahuje dvě instance třídy `Player`, která slouží k uložení informací o hráči. Ukládanými informacemi jsou typ hráče, tedy zda se jedná o lidského hráče nebo umělou inteligenci, jméno hráče či cesta k umělé inteligenci, informace o umělé inteligenci a dostupné nastavení umělé inteligence a jeho hodnoty. Nastavení a informace o umělé inteligenci jsou ukládány v asociativním poli stejně jako ve třídě `Tab` (viz **Chyba! Nenalezen zdroj odkazů. - Chyba! Nenalezen zdroj odkazů.**). Klíčem je tedy název ukládané informace a hodnotou je její aktuální hodnota.

5.2.3 Hrací plocha

Třída `Deska` slouží k uchování stavu hrací plochy, kontrole a provádění tahů a ukládání historie tahů. Její konkrétní implementace záleží na dané hře. Dá se však říci, že bude vždy obsahovat dvojrozměrné pole pro uložení současného stavu hry, určitou strukturu pro uchování historie tahů a několik funkcí pro získávání tahů a jejich kontrolu.

Pro předávání tahů se používají funkce `onSelected` a `onTurn`. První z uvedených slouží pro předání vybraného pole lidským hráčem. K provedení jednoho tahu tak může být potřeba vybrání několika polí. V této funkci je testováno, zda vybrané pole může být součástí platného tahu. Je-li po vybrání aktuálního pole tah kompletní, je volána funkce `onTurn`, které se předá uložená posloupnost vybraných polí. Pro předání tahu od umělé inteligence se využívá přímo funkce `onTurn`, jelikož umělá inteligence předává již kompletní tah. Ve funkci `onTurn` se zkontroluje, zda může být předaný tah proveden. Jestliže ano, je tah proveden a změny na hrací desce uloženy do historie tahů. Po úspěšném provedení tahu je vyvolán signál `sPlaced` do třídy `Plugin`. Tento signál obsahuje aktuální tah a slouží k předání provedeného tahu dalším umělým inteligencím.

Historie je ukládána od začátku hry a obsahuje všechny změny na hrací desce po provedení tahu daným hráčem. Pro některé hry, například piškvorky, by pro historii stačil seznam souřadnic, kdy každý pár souřadnic by značil jeden provedený tah. Použitý způsob uložení historie je však

použitelný pro kteroukoliv hru, jelikož počítá s množstvím změn na hrací desce po provedení jediného tahu. Pro uložení historie tahů jsou tedy využity následující struktury.

Do struktury `tChange` ukládáme změnu provedenou na hrací ploše. Seznamem těchto struktur získáme všechny změny provedené jedním tahem hráče.

```
typedef struct {
    int x;
    int y;
    int val;
} tChange;
typedef QList<tChange> tChanges;
```

Pro uložení konkrétního tahu provedeného hráčem je definována struktura `tMove` obsahující seznam změn a hráče, jenž daný tah provedl.

```
typedef struct{
    tChanges zmeny;
    int hrac;
}tMove;
```

Kompletní historii pak získáme jako seznam jednotlivých tahů.

```
QList<tMove> history;
```

Pro provedení kroku zpět zde nalezneme funkci `takeBack`, jež na hrací ploše vrátí provedené změny až po tah lidského hráče a tyto změny také vrátí zpět do třídy `Plugin`, odkud je volána, pro předání změn do umělých inteligencí. Funkce provádí krok zpět až k tahu lidského hráče, jelikož provedení jen jednoho kroku zpět by při hře s umělou inteligencí mělo za následek okamžité provedení stejného tahu umělou inteligencí. Z předchozí věty vyplývá, že krok zpět má smysl, pouze účastní-li se hry alespoň jeden lidský hráč.

Tato třída obsahuje také funkce `exportData` a `importData`, které jsou používány pro export a import dat při ukládání a načítání hry.

5.2.4 Vykreslování

`Painter` je třída starající se o vykreslování hrací desky a zpracování událostí kliknutí myší. Obsahuje ukazatele na třídy obsahující hrací plochu, nastavení, ladící informace a ukazatel na kreslící plátno. Z třídy `Settings` získává například informace o rozměrech hrací plochy, kterou bude vykreslovat. Třídu hrací plochy využívá pro získání hodnot jednotlivých polí desky k jejímu správnému vykreslení. Třída `QGraphicsView` je třída frameworku Qt, která funguje jako kreslící plátno. Tento ukazatel obsahuje adresu kreslícího plátna z třídy `Tab`, aby se o vykreslování mohla starat přímo knihovna hry pomocí této knihovny `Painter`. Podobně ukazatel na třídu `Debug` obsahuje adresu třídy `Debug` z `Tab`, kde jsou uloženy ladící informace určené k vykreslení.

Tato třída obsahuje dvě důležité funkce pro vykreslování hrací desky. První je funkce `paint`, která zkontroluje, zda nastavená šířka vykreslené scény odpovídá šířce kreslícího plátna. Pokud ne, je

volána druhá funkce `drawAll` pro překreslení hrací desky. Funkce `paint` je volána vždy, přijde-li z aplikace signál, že došlo k překreslení. Smyslem této funkce je překreslování scény, jen pokud dojde ke změně velikosti kreslícího plátna změnou velikosti okna aplikace.

Funkce `drawAll` je funkce pro vykreslení hrací desky, kde se na začátku funkce zjistí rozměry kreslícího plátna a podle počtu polí hrací desky se spočítají rozměry jednotlivých polí. Následně dojde k vykreslení hrací desky vložením obrázků na příslušné pozice na hrací desce, nebo vykreslením jednoduchých geometrických útvarů. Nakonec je zjištěno, zda je povoleno zobrazení ladících informací, a pokud ano, je volána funkce `drawDebug` pro vykreslení číselných ohodnocení z aktuálního zanoření do příslušných polí hrací desky. Tato funkce `drawAll` je volána z třídy `Plugin` po všech akcích, které mohou nějakým způsobem změnit zobrazení hrací desky.

Třída `Painter` také obsahuje funkci `onClicked` pro zpracování události kliknutí tlačítka myši na kreslící plátno. Při kliknutí tlačítka myši jsou zjištěny souřadnice kliknutí a dle počtu polí hrací desky a velikosti těchto polí jsou vypočteny souřadnice pole, na které bylo kliknuto. Poté se zjistí, zda šlo o kliknutí pravým nebo levým tlačítkem myši. Šlo-li o levé tlačítko myší, dojde k vyvolání signálu `sSelect`, jež je následně předán do třídy `Deska` a zpracován jako výběr pole lidským hráčem. Jednalo-li se o pravé tlačítko myši, volá se funkce `dbgClicked`, pomocí které se vykreslí další zanoření ladících informací, nebo se vybere základní úroveň zanoření. V této funkci se zjistí, zda je vybrané pole ohodnocené. Pokud ano, pokusí se zanořit do další úrovně, tedy získat ukazatel na ohodnocení v další úrovni (jak bylo popsáno v 4.2.4 - Ladící informace). Jde-li o pole bez ohodnocení, dojde k načtení základní úrovně ohodnocení, tedy získání ukazatele na základní úroveň ohodnocení. Po změně ukazatele na úroveň zanoření dojde k volání funkce `drawAll` a tím ke změně zobrazených ladících informací.

5.2.5 Rodičovská třída komunikačních protokolů

Třída `Brain` slouží jako rodičovská třída pro třídy jednotlivých komunikačních protokolů. Tato třída obsahuje ukazatel na proces umělé inteligence, číslo hráče, za kterého hraje, a cestu k aplikaci umělé inteligence. Obsahuje také funkci `write` pro zápis příkazu do procesu a funkce pro zjištění, zda proces běží, pro zjištění cesty k aplikaci a zjištění čísla hráče.

Dále třída obsahuje virtuální funkce definující rozhraní, které musí implementovat třídy komunikačních protokolů. Tyto funkce se mohou u různých her lišit. Mezi funkce, které se pravděpodobně vyskytnou u všech her, patří funkce pro začátek nové hry, nastavení hráče, za kterého umělá inteligence bude hrát, předání tahu umělé inteligenci, začátek výpočtu tahu, předání změn při kroku zpět, předání stavu hracího pole při načtení uložené hry, pro povolení zasílání ladících informací a pro ukončení umělé inteligence. Dále také funkce pro získání dostupných nastavení umělé inteligence a jejich nastavení novými hodnotami a funkce pro získání informací o umělé inteligenci a autorovi.

Tyto virtuální funkce jsou pak definovány v třídách komunikačních protokolů, které hra bude podporovat. V těch pak funkce zapisují pomocí funkce `write` do procesu příkazy daných protokolů.

5.2.6 Komunikace s umělými inteligencemi

Třída `COMM` slouží pro spouštění aplikací umělých inteligencí a komunikaci s nimi. Obsahuje asociativní pole, kde klíčem je číslo hráče a hodnotou je instance třídy `Brain`.

Většina funkcí třídy `COMM` pouze předává povely procesům umělých inteligencí, pokud existují. Nejdůležitější funkcí této třídy je zjištění, který protokol je podporován nově otevíranou umělou inteligencí.

Při spouštění nové umělé inteligence je volána funkce `openBrain`. Ta vytvoří nový proces se zadanou cestou k aplikaci umělé inteligence. Poté zavolá funkci `findProtocol`, která postupně vyzkouší jednotlivé podporované protokoly, dokud nenajde podporovaný protokol. Test na podporu daného protokolu je prováděn zapsáním příkazů do procesu a porovnáním výstupů s očekávanými výstupy. Používá-li umělá inteligence podporovaný protokol, je do funkce `openBrain` vráceno označení protokolu a do asociativního pole s hráči je na pozici daného hráče vytvořena nová instance třídy nalezeného protokolu.

5.3 Umělé inteligence

V této podkapitole se podíváme na implementaci samotných umělých inteligencí. Díky použití rour při komunikaci lze umělou inteligenci implementovat v jakémkoli programovacím jazyce, jenž je schopný pracovat se standardním vstupem a výstupem. V našem případě jsme opět zvolili prostředí Nokia Qt.

Hlavní třídou aplikace umělé inteligence je třída `Brain`. Tato třída zaobaluje umělou inteligenci a její hlavní funkcí je realizace komunikace pomocí komunikačního protokolu UBP. Pro realizaci neblokujícího čtení ze standardního vstupu je používána jednoduchá třída `Read`, která běží v dalším vlákně, snaží se číst ze standardního vstupu a při načtení příkazu jej předává pomocí signálu `sInput` do třídy `Brain`.

Příchozí zpráva se dostane do funkce `onInput`, kde se z ní vyjme první slovo, které představuje příkaz. Pro mapování příkazů na funkce je použito asociativní pole, kde klíčem jsou textové řetězce obsahující příkazy a hodnotami jsou ukazatele na patřičné funkce. V každé funkci je pak ještě zkontrolována správnost příkazu pomocí regulárního výrazu nebo testem na rovnost řetězců.

Díky použití asociativního pole pro mapování příkazů na funkce lze v průběhu měnit příkazy, kterým brain rozumí. Po spuštění očekává brain příkaz s názvem protokolu, po jeho přijetí jej z pole odebere a vloží ukazatel na funkci nastavující verzi protokolu. Obdobně pak dojde ke kontrole hry.

Poté je volána funkce `setUbpCommands`, ve které se nastaví ukazatele na příslušné funkce pro známé příkazy protokolu.

Většina funkcí této třídy si ze zprávy vyjme požadované informace a předá je do patřičné funkce některé z dalších tříd umělé inteligence, která se již stará o výpočet dalšího tahu.

Po získání tahu umělé inteligence je tah zaslán zpět komunikačním protokolem a je-li povolen výpis ladících informací, jsou zaslány i ony. Pro jejich výpis je použita funkce `debug`, která data vnitřně uložena v typu `QVariant` převádí za pomocí funkce `parseList` do formátu JSON. Řetězec s těmito daty je pak zaslán další zprávou komunikačního protokolu UBP, tedy příkazem `DEBUG`.

Samotný výpočet dalšího tahu umělé inteligence je prováděn pomocí prohledávání stavového prostoru algoritmem MiniMax s Alfa-Beta ořezáváním (viz 3.2 - Alfa-beta ořezávání), případně algoritmem Negamax, který je v podstatě jiným zápisem algoritmu alfa-beta, ale neobsahuje větvení, kvůli rozlišení, zda se hledá minimum nebo maximum.

6 Testování

V rámci této práce obsahuje nástroj dvě implementované hry - piškvorky a kameny. Aplikace byla testována v obou těchto hrách.

Hra kameny byla testována s použitím testovací umělé inteligence používající protokol UBP. Základem této umělé inteligence je umělá inteligence použitá v programu GUX od Ing. Jana Kouřila, která byla vložena do kostry brainu pro protokol UBP.

Hra piškvorky v nástroji podporuje dva protokoly - protokol UBP a protokol Petra Laštovičky. Byla testována na umělých inteligencích používajících oba dva tyto protokoly. Protokol UBP byl testován na vlastní testovací umělé inteligenci. Protokol Petra Laštovičky byl testován na různých umělých inteligencích, které se v minulosti účastnily piškvorkového turnaje na http://gomocup.wz.cz.

Testování všech možných kombinací hráčů bylo u obou her úspěšné. Nástroj tedy zvládá hru dvou lidských hráčů, hru jednoho lidského hráče a jedné umělé inteligence, i se začínající umělou inteligencí, a také hru dvou umělých inteligencí proti sobě, a to i takových, které používají odlišný komunikační protokol.

Během testování byl odhalen jeden problém týkající se umělých inteligencí pro protokol Petra Laštovičky. Některé z inteligencí zasílaly tah mimo hrací plochu a ukončily se, když při výpočtu tahu zjistily, že se již nemohou vyhnout prohře. Tento problém byl následně opraven kontrolou souřadnic tahu a zajištěním běžící umělé inteligence při startu hry.

7 Závěr

V rámci bakalářské práce byl navržen univerzální komunikační protokol, umožňující komunikaci umělých inteligencí a grafické části hry u různých typů her, a nástroj pro hraní her. Hlavní předností tohoto nástroje je možnost zobrazovat ladící informace z umělých inteligencí, a tím usnadnit programátorovi umělé inteligence kontrolu, zda se vyvíjená umělá inteligence chová tak, jak předpokládá. Možnost hry dvou umělých inteligencí proti sobě navíc umožní jejich porovnání. Další předností tohoto nástroje je použití dynamických knihoven pro implementované hry, což umožní uživatelům snadné přidávání nových her pouhým přidáním nové knihovny s hrou.

V budoucnu může být nástroj rozšířen o další podporované hry a možnost připojovat umělé inteligence přes další používané protokoly či také jiným způsobem, než přes roury, například přes TCP/IP. Vzhledem k tomu, že nástroj může kromě vývoje umělých inteligencí sloužit také čistě pro zábavu, tedy hraní různých her v jedné aplikaci, dalším možným rozšířením může být síťová dvou hráčů.

Literatura

[1] petr.lastovicka.sweb.cz: Piškvorky, protokol [online]. [cit. 2012-4-19].
Dostupné na URL: < http://petr.lastovicka.sweb.cz/protocl2.htm>

[2] gnu.org: Chess Engine Communication Protocol [online]. [cit. 2012-4-19].
Dostupné na URL: <http://www.gnu.org/software/xboard/engine-intf.html>

[3] wbec-ridderkerk.nl:Universal Chess Interface [online]. [cit. 2012-4-19].
Dostupné na URL: <http://wbec-ridderkerk.nl/html/UCIProtocol.html>

[4] cs.unm.edu : Game Tree Searching and pruning [online]. [cit. 2012-4-19].
Dostupné na URL: <http://www.cs.unm.edu/~aaron/downloads/qian_search.pdf>

[5] cs.cornell.edu: Minimax search and Alpha-Beta Pruning [online]. [cit. 2012-4-19].
Dostupné na URL: < http://www.cs.cornell.edu/courses/cs312/2002sp/lectures/rec21.htm>

[6] Zbořil F.: Základy umělé inteligence, studijní opora, VUT FIT, 2006 [cit. 2012-4-19].

Seznam příloh

Příloha A - Protokol pro piškvorky

Stručný popis příkazů protokolu pro piškvorky od Petra Laštovičky. Kompletní popis protokolu je dostupný na http://petr.lastovicka.sweb.cz/protocl2.htm.

Povinné příkazy

START [velikost] - Příkaz sloužící k inicializaci brainu. Parametr velikost určuje velikost čtvercové hrací plochy. Pokud brain velikost podporuje, odpoví OK, jinak odpoví **ERROR [chybová hláška].**

TURN [X],[Y] - Říká brainu, na které souřadnice táhl soupeř. Souřadnice jsou číslovány od nuly. Brain odpoví souřadnicemi svého tahu, tedy dvěma čísly oddělenými čárkou.

BEGIN - Brain, které tento příkaz dostane, ví, že má začít na volné ploše hrát. Odpoví stejně jako po příkazu **TURN**, tedy souřadnicemi svého tahu.

BOARD - Příkaz pro vnucení herní plochy. Slouží například pro pokračování v rozehrané partii. Po tomto příkazu následují příkazy **[X],[Y],[POLE]**, kde X a Y jsou souřadnice a POLE je buď číslo 1 (vlastní piškvorka) nebo 2 (piškvorka soupeře), nebo 3 (výherní řada při kontinuální hře). Data jsou ukončena samostatným příkazem **DONE**. Po tomto příkazu se od brainu očekává odpověď jako po příkazu **TURN** nebo **BEGIN.**

INFO [klíč] [hodnota] - Příkaz sloužící pro předání brainu určité informace. Parametr [klíč] určuje, jaká informace se bude předávat. Parametrem mohou být následující hodnoty: timeout_turn (limit na tah), timeout_match (limit na zápas), max_memory (limit na maximální spotřebovanou paměť), time_left (zbývající čas do konce partie), game_type (hra proti člověku/brainu/turnaj/síťový turnaj), rule (výhra při 5 a více/přesně 5 v řadě; normální/kontinuální hra), evaluate (souřadnice myši) a folder (složka pro zápis trvalých souborů).

END - Příkaz k co nejrychlejšímu ukončení brainu.

ABOUT - Získání informací o brainu. Brain na jedné řádce pošle informace o svém názvu, autorovi, verzi a další.

Nepovinné příkazy

RECTSTART [šířka],[výška] - Příkaz pro start hry na obdélníkové hrací ploše. Brain odpoví **OK**, pokud zadanou obdélníkovou velikost podporuje, **ERROR** v případě, že ne.

RESTART - Příkaz pro novou inicializaci hrací plochy se současným nastavením. Brain po restartu odpoví **OK.**

TAKEBACK [X],[Y] - Příkaz pro odebrání piškvorky ze zadaných souřadnic. Slouží například pro krok zpět. Brain odpoví **OK.**

PLAY [X],[Y] - Vnucení tahu brainu. Manažer může brainu jako odpověd na **SUGGEST** vnutit jiný tah, než brain navrhl.

Příkazy, které posílá brain

Následující příkazy může použít brain pokud je potřebuje.

UNKNOWN [chybová zpráva] - Reakce brainu na příchozí příkaz, který brain nezná.

ERROR [chybová zpráva] - Reakce brainu na příkaz, který sice zná, ale nemůže jej provést.

MESSAGE [zpráva] - Zpráva pro logování do souboru nebo výpis do nějakého okna.

DEBUG [zpráva] - Stejné jako **MESSAGE**, ale zpravidla obsahuje informace, kterým rozumí jen autor brainu.

SUGGEST [X],[Y] - Rozšíření protokolu pro debugování a porovnávání brainů. Pokud brain před souřadnice tahu umístí příkaz **SUGGEST**, nemění svůj vnitřní stav a dává manažeru najevo, že má pomocí **PLAY** tah potvrdit, nebo mu vnutit jiný.

Příloha B - Chess Engine Communication Protocol

Stručný popis příkazů protokolu Chess Engine Communication Protocol pro hru šachy. Vzhledem k velkému množství příkazů je uvedena pouze jejich část, potřebná pro vytvoření a průběh hry. Kompletní popis protokolu je dostupný na http://www.gnu.org/software/xboard/engine-intf.html.

Příkazy

xboard - Příkaz zaslaný brainu ihned po jeho spuštění. Slouží pro nastavení brainu na používání právě tohoto protokolu.

protover [N] - Nastavení verze protokolu. Není-li tento příkaz zaslán, předpokládá se verze 1.

new - Nastavení herní plochy do standardní startovní pozice a reset všech nastavení z předchozích her.

variant [VARNAME] - Nastavení jiné varianty hry než standardní šachy.

quit - Příkaz pro okamžité ukončení brainu.

st [TIME] - Nastavení času

sd [DEPTH] - Nastavení hloubky přemýšlení

[MOVE] - Předání tahu od manažera brainu - předává se pouze tah, bez příkazu. Brain začne přemýšlet a odpoví vlastním tahem.

move [MOVE] - Zpráva od brainu s informací o tahu, který táhne.

Illegal move [(REASON)]: [MOVE] - Zpráva od brainu, která informuje grafické rozhraní o provedení neplatného tahu. Je možnost uvést důvod, proč tah nelze provést.

Error [(ERRORTYPE)]: [COMMAND] - Ohlášení chyby, pokud brain příkaz nezná nebo jej neimplementuje.

? - Příkaz pro okamžitý tah brainu pokud přemýšlí.

ping [N] - Dotaz na brain, zda již dokončil předchozí příkazy. Jakmile jsou předchozíí příkazy zpracovány, brain odpoví **pong [N].**

draw - Nabídnutí remízy. Pokud brain souhlasí, odpoví **offer draw**, jinak příkaz ignoruje.

result [RESULT] {[COMMENT]} - Zaslání výsledku po skončení hry.

edit - Příkaz pro přepnutí brainu do editačního módu. V editačním módu jsou dostupné následující příkazy: „c“ - změna barvy (defaultně bílá), „#“ - smazat plochu, „.“ - opustit editační mód, „Pa4“ - příklad vložení pěšáka na a4, „xa4“ - smazat figurku z a4

undo - Příkaz pro krok zpět v případě, že brain nepřemýšlí. Vrátí zpět hráčův tah.

remove - Příkaz pro krok zpět. Brain vezme zpět dva tahy - svůj a hráčův.

hard/easy - Zapnutí/vypnutí přemýšlení během tahu druhého hráče.

memory [N] - Nastavení omezení paměti brainu.

cores [N] - Omezení počtu jader použitých brainem.

option [NAME]=[VALUE] - Příkaz pro nastavení hodnoty NAME na hodnotu VALUE. Slouží pro nastavování různých hodnot

Rozšíření

Od verze protokolu 2 je možné, aby brain zaslal manažeru seznam proměnných ve tvaru Název=HODNOTA, pomocí kterých může manažeru předat informace o tom, které rozšiřující funkce a příkazy implementuje. Tímto způsobem také může ovlivňovat vzhled nebo chování manažeru.

Předávání těchto hodnot se provádí příkazem

feature [FEATURE1]=[VALUE1] [FEATURE2]=[VALUE2] …
Brain tyto hodnoty zasílá ihned po obdržení příkazu **protover** s verzí větší rovno 2. Vzhledem k tomu, že tyto proměnné či příkazy nejsou implementovány povinně a mohou být neustále přidávány, je nutné, aby byly potvrzovány manažerem v případě, že jim rozumí.

Přijetí a odmítnut zaslaného příkazu **feature** probíhá pomocí příkazů **accepted** a **rejected**.

Příloha C - Universal Chess Interface

Stručný popis příkazů protokolu Universal Chess Interface pro hru šachy. Kompletní popis protokolu je dostupný na http://wbec-ridderkerk.nl/html/UCIProtocol.html.

Příkazy z GUI

uci - Zpráva pro brain, že má používat UCI protokol. Tento příkaz by měl být poslán jako první příkaz pro brain. Ten na něj odpoví příkazy **id** a **option**, sloužícími pro identifikaci a předání nastavení brainu. Poté je zaslán příkaz **uci ok** jako potvrzení použití protokolu UCI.

debug [on | off] - Zapnutí/vypnutí ladícího režimu. Pomocné ladící informace jsou pak zasílány brainem pomocí příkazu **info.**

isready - Příkaz sloužící pro synchronizaci grafického rozhraní s brainem. Tento příkaz může být zaslán po jednom nebo více příkazech, u kterých se očekává, že jejich zpracování bude trvat delší dobu. Jakmile je brain připraven, odpoví **readyok.**

setoption name [name] value [value] - Příkaz sloužící ke změně vnitřních nastavení brainu.

register - Příkaz sloužící pro registraci brainu. Zaslán jako reakce na zprávu od brainu **registration error**. Příkaz může obsahovat jméno nebo kód pro registraci (nebo obojí), případně informaci **later** o pozdější registraci.

ucinewgame - Příkaz informující o tom, že další vyhledání tahu se bude vztahovat k jiné hře.

position [fen <fenstring>| startpos] moves <move1> … <movei> - Příkaz pro rozestavení figur na interním hracím poli podle řetězce „fenstring“ a odehrání tahů popsaných v „moves“. Pokud je rozestavení figur rovno počátečnímu rozestavení, je možno použít „startpos“.

go - Příkaz pro zahájení přemýšlení nad aktuální pozicí. Za tento příkaz mohou být připojeny další příkazy. Ty mohou informovat o zbývajícím čase hráčů nebo omezovat hodnoty jako čas pro přemýšlení, hloubku přemýšlení, počet prohledaných uzlů a další.

stop - Příkaz pro co nejrychlejší ukončení přemýšlení.

ponderhit - Oznámení brainu, že hráč hrál předpokládaný tah a tím může brain pokračovat v již rozpracovaném výpočtu.

quit - Příkaz pro okamžité ukončení brainu.

Příkazy z brainu

id - Příkaz sloužící k identifikaci brainu. Musí být zaslán po obdržení příkazu **uci.** Brain se identifikuje pomocí jména a autora, tedy příkazy **id name [name]** a **id author [author]**

uciok - Příkaz pro oznámení, že brain již poslal všechny potřebné informace, používá režim uci a je připraven. Je zasílán po příkazech **id** a volitelném **option.**

readyok - Tento příkaz musí brain poslat po obdržení příkazu **isready** poté, co zpracuje všechny vstupy a je přípraven pro zpracování dalších příkazů.

bestmove [move] - Příkaz vracející nejlepší nalezený tah po tom, co brain přestal počítat.

copyprotection - Příkaz používaný brainy s ochranou proti kopírování. Po příkazu **uciok** může brain zaslat příkaz **copyprotection checking** a poté **copyprotection ok** nebo **copyprotection error** jako informaci o úspěšném resp. neúspěšném výsledku kontroly.

registration - Příkaz potřebný pro brainy, které vyžadují uživatelské jméno a/nebo kód pro povolení všech jejich funkcí. Pracuje stejně jako **copyprotection**, tedy se stavy **checking**, **ok** a **error**.

info - Příkaz sloužící pro zaslání informací z brainu do grafického rozhraní hry. Tento příkaz by měl být poslán kdykoliv se nějaká informace změní. Informace mohou být posílány jednotlivými příkazy, nebo jako více informací v jednom příkazu. Předávané informace se mohou týkat hloubky prohledávání, času hledání, počtu prohledaných uzlů, rychlosti prohledávání, zatížení procesoru a dalších.

option - Příkaz zasílaný po startu brainu, po obdržení příkazu **uci**. Slouží k informování grafického rozhraní hry o parametrech, které lze v brainu měnit. Chce-li grafické rozhraní hry nějaké parametry změnit z jejich defaultní hodnoty, učiní tak pomocí příkazu **setoption.**

Příloha D - Manuál

Instalace

Pro snadné použití je binární podoba aplikace přiložena jako instalace. Tato instalace je dostupná pro platformy Windows, Linux 32-bit a Linux 64-bit. Instalace probíhá formou grafického průvodce instalací a zajistí instalaci potřebných dynamických knihoven nutných pro spuštění programu. V případě problémů s instalací je přiložen také archiv s binárními soubory aplikace, avšak při použití této možnosti musí potřebné dynamické knihovny zajistit sám uživatel.
Při instalaci je možné vybrat, zda se mají nainstalovat také testovací umělé inteligence pro implementované hry. Je-li tato možnost povolena, jsou tyto umělé inteligence umístěny do podsložky brains/ v kořenovém adresáři instalace.

Spuštění aplikace

V prostředí Windows se spuštění provádí spuštěním aplikace AITool.exe. V prostředí Linux se aplikace spouští pomocí skriptu run.sh, který zajistí použití potřebných dynamických knihoven.

Spuštění hry

K výběru požadované hry dojde vybráním příslušné záložky s názvem hry. Poté je možné nastavit hráče hry. K nastavení hráče jako umělé inteligence dojde nastavením příslušného typu hráče a následně výběrem požadované umělé inteligence. K samotnému zahájení hry dojde stisknutím příslušného tlačítka. V aplikaci může běžet současně více her na více záložkách.

Přidání dalších her

Hry zobrazené v panelu záložek jsou automaticky při spuštění aplikace načítány ze složky plugins/ v adresáři aplikace. Další hru je možné přidat vložením příslušné dynamické knihovny hry do uvedené složky a restartováním aplikace.

Změna jazyka

Aplikace je lokalizována v českém a anglickém jazyce. Výběr jazyka je možný z menu Možnosti. Ke změně jazyka dojde při opětovném spuštění aplikace.